Deep Learning Model

Model Pseudocode

In [ ]:
```
# Function create_cnn_model(input_shape):

#     Initialize Sequential model

#     Add Conv1D(32, kernel=3, padding='same', input_shape)
#     Add BatchNormalization
#     Add LeakyReLU(α=0.3)
#     Add MaxPooling1D(pool=2)

#     Add Conv1D(64, kernel=3, padding='same')
#     Add BatchNormalization
#     Add LeakyReLU(α=0.3)
#     Add MaxPooling1D(pool=2)

#     Add Conv1D(128, kernel=3, padding='same')
#     Add BatchNormalization
#     Add LeakyReLU(α=0.3)

#     Add GlobalAveragePooling1D

#     Add Dense(64) → BatchNormalization → LeakyReLU(α=0.3)
#     Add Dropout(0.25)

#     Add Output Dense(1, activation='sigmoid')

#     Compile model with:
#         optimizer = 'adam'
#         loss = 'binary_crossentropy'
#         metrics = ['accuracy']

#     Return model
```

Step 1: Load and Prepare the Dataset

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import MinMaxScaler

# Load dataset
df = pd.read_csv('D:\Coding Projects\Detection-of-SYN-Flood-Attacks-Using-Machine-L
X = df.drop('Label', axis=1).values
y = df['Label'].values
X = X.reshape(X.shape[0], X.shape[1], 1)

# Flatten before scaling and reshape after
X_flat = X.reshape(X.shape[0], X.shape[1])
```

```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_flat)

# Reshape back to 3D for CNN input
X = X_scaled.reshape(X.shape[0], X.shape[1], 1)
```

Step 2: Defining the 1D CNN Architecture

In [2]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, D

def create_cnn_model(input_shape):
    model = Sequential()

    model.add(Conv1D(32, kernel_size=3, padding='same', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.3))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(64, kernel_size=3, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.3))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(128, kernel_size=3, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.3))

    model.add(GlobalAveragePooling1D())

    model.add(Dense(64))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.3))
    model.add(Dropout(0.25))  # Moderate regularization

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'
    return model


##Initttaaallll Modelll
# from tensorflow.keras.models import Sequential
# from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

# def create_improved_cnn_model(input_shape):
#     model = Sequential([
#         Conv1D(64, kernel_size=3, activation='relu', input_shape=input_shape),
#         BatchNormalization(),
#         MaxPooling1D(pool_size=2),

#         Conv1D(128, kernel_size=3, activation='relu'),
#         BatchNormalization(),
#         MaxPooling1D(pool_size=2),
```

```
#         Conv1D(256, kernel_size=3, activation='relu'),
#         BatchNormalization(),

#         Flatten(),
#         Dense(128, activation='relu'),
#         Dropout(0.3),  # Less aggressive than 0.5
#         Dense(64, activation='relu'),
#         Dropout(0.3),
#         Dense(1, activation='sigmoid')  # Binary classification
#     ])

#     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accurac
#     return model
```

Step 3: Training the Model using the K-Folds + Resource Management

In [11]:

```
import time
import psutil
import os

accuracies = []
all_y_true = []
all_y_pred = []
all_y_scores = []

process = psutil.Process(os.getpid())

# Resource Monitoring Start
overall_start_time = time.time()
overall_start_ram = process.memory_info().rss / 1024 / 1024  # in MB
overall_start_cpu = psutil.cpu_percent(interval=1)

for fold in range(0, 5):
    print(f"\n--- Training on Fold {fold} ---")

    train_idx = df[df['Fold'] != fold].index
    test_idx = df[df['Fold'] == fold].index

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    model = create_cnn_model(input_shape=X.shape[1:])

    # Train Model
    history = model.fit(
        X_train, y_train,
        epochs=30,
        batch_size=64,
        validation_data=(X_test, y_test),
        verbose=1
    )

    #  Evaluation
    y_scores = model.predict(X_test).ravel()
```

```
    y_pred = (y_scores > 0.5).astype(int)

    all_y_true.extend(y_test)
    all_y_pred.extend(y_pred)
    all_y_scores.extend(y_scores)

    loss, acc = model.evaluate(X_test, y_test, verbose=0)
    accuracies.append(acc)

# Resource Monitoring End
overall_end_time = time.time()
overall_end_ram = process.memory_info().rss / 1024 / 1024  # in MB
overall_end_cpu = psutil.cpu_percent(interval=1)

# Summary
print("\n Overall Training Stats ")
print(f"Total Training Time: {overall_end_time - overall_start_time:.2f} seconds")
print(f"Total RAM Usage Increase: {overall_end_ram - overall_start_ram:.2f} MB")
print(f"CPU Usage (at final check): {overall_end_cpu}%")
```

```
--- Training on Fold 0 ---
Epoch 1/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9562 - loss: 0.1067 - val_acc
uracy: 0.5154 - val_loss: 1.0641
Epoch 2/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0152 - val_acc
uracy: 0.5154 - val_loss: 1.0457
Epoch 3/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9944 - loss: 0.0313 - val_acc
uracy: 0.5154 - val_loss: 0.8478
Epoch 4/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0180 - val_acc
uracy: 0.9990 - val_loss: 0.1203
Epoch 5/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0104 - val_acc
uracy: 1.0000 - val_loss: 0.0211
Epoch 6/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9975 - loss: 0.0112 - val_acc
uracy: 0.9995 - val_loss: 0.0027
Epoch 7/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0086 - val_acc
uracy: 1.0000 - val_loss: 0.0043
Epoch 8/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0090 - val_acc
uracy: 1.0000 - val_loss: 0.0051
Epoch 9/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9885 - loss: 0.0394 - val_acc
uracy: 0.9990 - val_loss: 0.0029
Epoch 10/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0098 - val_acc
uracy: 1.0000 - val_loss: 0.0020
Epoch 11/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9977 - loss: 0.0098 - val_acc
uracy: 1.0000 - val_loss: 0.0016
Epoch 12/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0080 - val_acc
uracy: 1.0000 - val_loss: 0.0015
Epoch 13/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0070 - val_acc
uracy: 1.0000 - val_loss: 0.0016
Epoch 14/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9942 - loss: 0.0219 - val_acc
uracy: 0.9990 - val_loss: 0.0036
Epoch 15/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0083 - val_acc
uracy: 1.0000 - val_loss: 0.0152
Epoch 16/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9960 - loss: 0.0264 - val_acc
uracy: 1.0000 - val_loss: 0.0050
Epoch 17/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0130 - val_acc
uracy: 0.9990 - val_loss: 0.0031
Epoch 18/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9971 - loss: 0.0121 - val_acc
uracy: 1.0000 - val_loss: 0.0019
Epoch 19/30
```

```
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9989 - loss: 0.0078 - val_acc
uracy: 1.0000 - val_loss: 0.0033
Epoch 20/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0101 - val_acc
uracy: 1.0000 - val_loss: 0.0011
Epoch 21/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0091 - val_acc
uracy: 0.9990 - val_loss: 0.0017
Epoch 22/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9971 - loss: 0.0130 - val_acc
uracy: 1.0000 - val_loss: 0.0011
Epoch 23/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0077 - val_acc
uracy: 0.9974 - val_loss: 0.0039
Epoch 24/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0062 - val_acc
uracy: 1.0000 - val_loss: 0.0089
Epoch 25/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9989 - loss: 0.0056 - val_acc
uracy: 1.0000 - val_loss: 8.7412e-04
Epoch 26/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0074 - val_acc
uracy: 0.9995 - val_loss: 8.6630e-04
Epoch 27/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9989 - loss: 0.0052 - val_acc
uracy: 1.0000 - val_loss: 5.9775e-04
Epoch 28/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0064 - val_acc
uracy: 0.9995 - val_loss: 0.0011
Epoch 29/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.0122 - val_acc
uracy: 0.9995 - val_loss: 8.9011e-04
Epoch 30/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0054 - val_acc
uracy: 1.0000 - val_loss: 5.9921e-04
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

--- Training on Fold 1 ---
Epoch 1/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9651 - loss: 0.0872 - val_acc
uracy: 0.5154 - val_loss: 1.3674
Epoch 2/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9974 - loss: 0.0158 - val_acc
uracy: 0.5154 - val_loss: 1.6655
Epoch 3/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0085 - val_acc
uracy: 0.5154 - val_loss: 1.0243
Epoch 4/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0094 - val_acc
uracy: 0.9984 - val_loss: 0.0683
Epoch 5/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9974 - loss: 0.0094 - val_acc
uracy: 0.9995 - val_loss: 0.0028
Epoch 6/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9974 - loss: 0.0086 - val_acc
uracy: 0.9995 - val_loss: 0.0026
```

```
Epoch 7/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0075 - val_acc
uracy: 0.9995 - val_loss: 0.0012
Epoch 8/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0067 - val_acc
uracy: 0.9995 - val_loss: 0.0022
Epoch 9/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.0121 - val_acc
uracy: 0.9995 - val_loss: 0.0016
Epoch 10/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0098 - val_acc
uracy: 0.9990 - val_loss: 0.0024
Epoch 11/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9992 - loss: 0.0058 - val_acc
uracy: 0.9995 - val_loss: 0.0019
Epoch 12/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0076 - val_acc
uracy: 0.9984 - val_loss: 0.0027
Epoch 13/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9972 - loss: 0.0107 - val_acc
uracy: 0.9974 - val_loss: 0.0154
Epoch 14/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9911 - loss: 0.0258 - val_acc
uracy: 0.9995 - val_loss: 0.0021
Epoch 15/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9966 - loss: 0.0153 - val_acc
uracy: 0.9984 - val_loss: 0.0043
Epoch 16/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9978 - loss: 0.0104 - val_acc
uracy: 1.0000 - val_loss: 0.0018
Epoch 17/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0086 - val_acc
uracy: 0.9990 - val_loss: 0.0027
Epoch 18/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9969 - loss: 0.0142 - val_acc
uracy: 0.9990 - val_loss: 0.0022
Epoch 19/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0081 - val_acc
uracy: 0.9969 - val_loss: 0.0234
Epoch 20/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9923 - loss: 0.0272 - val_acc
uracy: 0.9990 - val_loss: 0.0029
Epoch 21/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0085 - val_acc
uracy: 0.9990 - val_loss: 0.0030
Epoch 22/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0059 - val_acc
uracy: 0.9990 - val_loss: 0.0020
Epoch 23/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0094 - val_acc
uracy: 0.9995 - val_loss: 0.0014
Epoch 24/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9989 - loss: 0.0067 - val_acc
uracy: 0.9995 - val_loss: 0.0020
Epoch 25/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0076 - val_acc
```

```
uracy: 0.9990 - val_loss: 0.0023
Epoch 26/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0152 - val_acc
uracy: 0.9990 - val_loss: 0.0033
Epoch 27/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0076 - val_acc
uracy: 0.9995 - val_loss: 0.0018
Epoch 28/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0089 - val_acc
uracy: 0.9995 - val_loss: 0.0014
Epoch 29/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0108 - val_acc
uracy: 0.9995 - val_loss: 0.0017
Epoch 30/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9994 - loss: 0.0033 - val_acc
uracy: 0.9995 - val_loss: 0.0021
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

--- Training on Fold 2 ---
Epoch 1/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9722 - loss: 0.0861 - val_acc
uracy: 0.5154 - val_loss: 1.1950
Epoch 2/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0251 - val_acc
uracy: 0.5154 - val_loss: 1.7875
Epoch 3/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9968 - loss: 0.0209 - val_acc
uracy: 0.5154 - val_loss: 0.5244
Epoch 4/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0115 - val_acc
uracy: 0.9969 - val_loss: 0.0762
Epoch 5/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss: 0.0045 - val_acc
uracy: 0.9974 - val_loss: 0.0159
Epoch 6/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0089 - val_acc
uracy: 0.9974 - val_loss: 0.0128
Epoch 7/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0070 - val_acc
uracy: 0.9974 - val_loss: 0.0143
Epoch 8/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0051 - val_acc
uracy: 0.9974 - val_loss: 0.0127
Epoch 9/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0078 - val_acc
uracy: 0.9969 - val_loss: 0.0135
Epoch 10/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0061 - val_acc
uracy: 0.9974 - val_loss: 0.0157
Epoch 11/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9937 - loss: 0.0252 - val_acc
uracy: 0.9927 - val_loss: 0.0225
Epoch 12/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9966 - loss: 0.0182 - val_acc
uracy: 0.9964 - val_loss: 0.0174
Epoch 13/30
```

```
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0104 - val_acc
uracy: 0.9969 - val_loss: 0.0174
Epoch 14/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9966 - loss: 0.0186 - val_acc
uracy: 0.9974 - val_loss: 0.0181
Epoch 15/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0072 - val_acc
uracy: 0.9974 - val_loss: 0.0155
Epoch 16/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0101 - val_acc
uracy: 0.9974 - val_loss: 0.0157
Epoch 17/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0072 - val_acc
uracy: 0.9974 - val_loss: 0.0152
Epoch 18/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0078 - val_acc
uracy: 0.9958 - val_loss: 0.0156
Epoch 19/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0097 - val_acc
uracy: 0.9974 - val_loss: 0.0140
Epoch 20/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9994 - loss: 0.0040 - val_acc
uracy: 0.9969 - val_loss: 0.0165
Epoch 21/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9992 - loss: 0.0047 - val_acc
uracy: 0.9974 - val_loss: 0.0148
Epoch 22/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0063 - val_acc
uracy: 0.9974 - val_loss: 0.0145
Epoch 23/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0069 - val_acc
uracy: 0.9974 - val_loss: 0.0176
Epoch 24/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9848 - loss: 0.0467 - val_acc
uracy: 0.9927 - val_loss: 0.0205
Epoch 25/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0120 - val_acc
uracy: 0.9974 - val_loss: 0.0148
Epoch 26/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9978 - loss: 0.0101 - val_acc
uracy: 0.9974 - val_loss: 0.0139
Epoch 27/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9973 - loss: 0.0105 - val_acc
uracy: 0.9969 - val_loss: 0.0146
Epoch 28/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0088 - val_acc
uracy: 0.9969 - val_loss: 0.0139
Epoch 29/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0117 - val_acc
uracy: 0.9974 - val_loss: 0.0129
Epoch 30/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss: 0.0143 - val_acc
uracy: 0.9974 - val_loss: 0.0139
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

--- Training on Fold 3 ---
```

```
Epoch 1/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9565 - loss: 0.1093 - val_acc
uracy: 0.5154 - val_loss: 1.1480
Epoch 2/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0090 - val_acc
uracy: 0.5154 - val_loss: 1.4117
Epoch 3/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.0082 - val_acc
uracy: 0.7543 - val_loss: 0.3598
Epoch 4/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0061 - val_acc
uracy: 0.9964 - val_loss: 0.2604
Epoch 5/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9939 - loss: 0.0209 - val_acc
uracy: 0.9969 - val_loss: 0.0422
Epoch 6/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0124 - val_acc
uracy: 0.9969 - val_loss: 0.0230
Epoch 7/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9987 - loss: 0.0054 - val_acc
uracy: 0.9964 - val_loss: 0.0501
Epoch 8/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0060 - val_acc
uracy: 0.9979 - val_loss: 0.0173
Epoch 9/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0043 - val_acc
uracy: 0.9974 - val_loss: 0.0185
Epoch 10/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0050 - val_acc
uracy: 0.9979 - val_loss: 0.0214
Epoch 11/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9958 - loss: 0.0164 - val_acc
uracy: 0.9974 - val_loss: 0.0198
Epoch 12/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9994 - loss: 0.0040 - val_acc
uracy: 0.9969 - val_loss: 0.0258
Epoch 13/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0085 - val_acc
uracy: 0.9979 - val_loss: 0.0204
Epoch 14/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0035 - val_acc
uracy: 0.9964 - val_loss: 0.0223
Epoch 15/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0042 - val_acc
uracy: 0.9979 - val_loss: 0.0202
Epoch 16/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9972 - loss: 0.0066 - val_acc
uracy: 0.9974 - val_loss: 0.0229
Epoch 17/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9980 - loss: 0.0101 - val_acc
uracy: 0.9974 - val_loss: 0.0198
Epoch 18/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9973 - loss: 0.0089 - val_acc
uracy: 0.9979 - val_loss: 0.0194
Epoch 19/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0051 - val_acc
```

```
uracy: 0.9979 - val_loss: 0.0209
Epoch 20/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0043 - val_acc
uracy: 0.9964 - val_loss: 0.0212
Epoch 21/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9992 - loss: 0.0033 - val_acc
uracy: 0.9964 - val_loss: 0.0239
Epoch 22/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9993 - loss: 0.0041 - val_acc
uracy: 0.9964 - val_loss: 0.0232
Epoch 23/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0031 - val_acc
uracy: 0.9979 - val_loss: 0.0229
Epoch 24/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9969 - loss: 0.0110 - val_acc
uracy: 0.9964 - val_loss: 0.0218
Epoch 25/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0066 - val_acc
uracy: 0.9974 - val_loss: 0.0241
Epoch 26/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9952 - loss: 0.0200 - val_acc
uracy: 0.9979 - val_loss: 0.0182
Epoch 27/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0041 - val_acc
uracy: 0.9964 - val_loss: 0.0210
Epoch 28/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9979 - loss: 0.0071 - val_acc
uracy: 0.9964 - val_loss: 0.0207
Epoch 29/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9913 - loss: 0.0270 - val_acc
uracy: 0.9969 - val_loss: 0.0227
Epoch 30/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0074 - val_acc
uracy: 0.9974 - val_loss: 0.0196
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

--- Training on Fold 4 ---
Epoch 1/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9575 - loss: 0.1147 - val_acc
uracy: 0.5151 - val_loss: 1.1434
Epoch 2/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0136 - val_acc
uracy: 0.5151 - val_loss: 1.5548
Epoch 3/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9958 - loss: 0.0160 - val_acc
uracy: 0.5151 - val_loss: 1.3015
Epoch 4/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0088 - val_acc
uracy: 0.5151 - val_loss: 0.6592
Epoch 5/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9958 - loss: 0.0260 - val_acc
uracy: 0.9979 - val_loss: 0.0185
Epoch 6/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9977 - loss: 0.0106 - val_acc
uracy: 0.9979 - val_loss: 0.0148
Epoch 7/30
```

```
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9974 - loss: 0.0100 - val_acc
uracy: 0.9979 - val_loss: 0.0141
Epoch 8/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0080 - val_acc
uracy: 0.9969 - val_loss: 0.0165
Epoch 9/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9969 - loss: 0.0124 - val_acc
uracy: 0.9979 - val_loss: 0.0117
Epoch 10/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9978 - loss: 0.0123 - val_acc
uracy: 0.9979 - val_loss: 0.0091
Epoch 11/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9983 - loss: 0.0072 - val_acc
uracy: 0.9979 - val_loss: 0.0078
Epoch 12/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9982 - loss: 0.0092 - val_acc
uracy: 0.9979 - val_loss: 0.0103
Epoch 13/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9975 - loss: 0.0119 - val_acc
uracy: 0.9979 - val_loss: 0.0085
Epoch 14/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0055 - val_acc
uracy: 0.9979 - val_loss: 0.0075
Epoch 15/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9978 - loss: 0.0093 - val_acc
uracy: 0.9964 - val_loss: 0.1354
Epoch 16/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9939 - loss: 0.0214 - val_acc
uracy: 0.9979 - val_loss: 0.0129
Epoch 17/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.0134 - val_acc
uracy: 0.9979 - val_loss: 0.0107
Epoch 18/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9977 - loss: 0.0108 - val_acc
uracy: 0.9979 - val_loss: 0.0082
Epoch 19/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0082 - val_acc
uracy: 0.9974 - val_loss: 0.0083
Epoch 20/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9976 - loss: 0.0121 - val_acc
uracy: 0.9979 - val_loss: 0.0072
Epoch 21/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0055 - val_acc
uracy: 0.9979 - val_loss: 0.0072
Epoch 22/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9992 - loss: 0.0040 - val_acc
uracy: 0.9979 - val_loss: 0.0070
Epoch 23/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9988 - loss: 0.0069 - val_acc
uracy: 0.9984 - val_loss: 0.0059
Epoch 24/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9985 - loss: 0.0048 - val_acc
uracy: 0.9984 - val_loss: 0.0051
Epoch 25/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss: 0.0048 - val_acc
uracy: 0.9979 - val_loss: 0.0278
```

```
Epoch 26/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9992 - loss: 0.0038 - val_acc
uracy: 0.9984 - val_loss: 0.0054
Epoch 27/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9981 - loss: 0.0095 - val_acc
uracy: 0.9885 - val_loss: 0.0368
Epoch 28/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0076 - val_acc
uracy: 0.9984 - val_loss: 0.0049
Epoch 29/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9984 - loss: 0.0062 - val_acc
uracy: 0.9984 - val_loss: 0.0045
Epoch 30/30
121/121 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9986 - loss: 0.0065 - val_acc
uracy: 0.9979 - val_loss: 0.0100
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 918us/step

 Overall Training Stats
Total Training Time: 59.94 seconds
Total RAM Usage Increase: 127.69 MB
CPU Usage (at final check): 4.3%
```

Step 4: Evaluation

In [9]:

```python
import numpy as np

print("\nFinal CNN Cross-Validation Results:")
print(f"Fold Accuracies: {accuracies}")
print(f"Mean Accuracy: {np.mean(accuracies):.4f}")
print(f"Standard Deviation: {np.std(accuracies):.4f}")
```

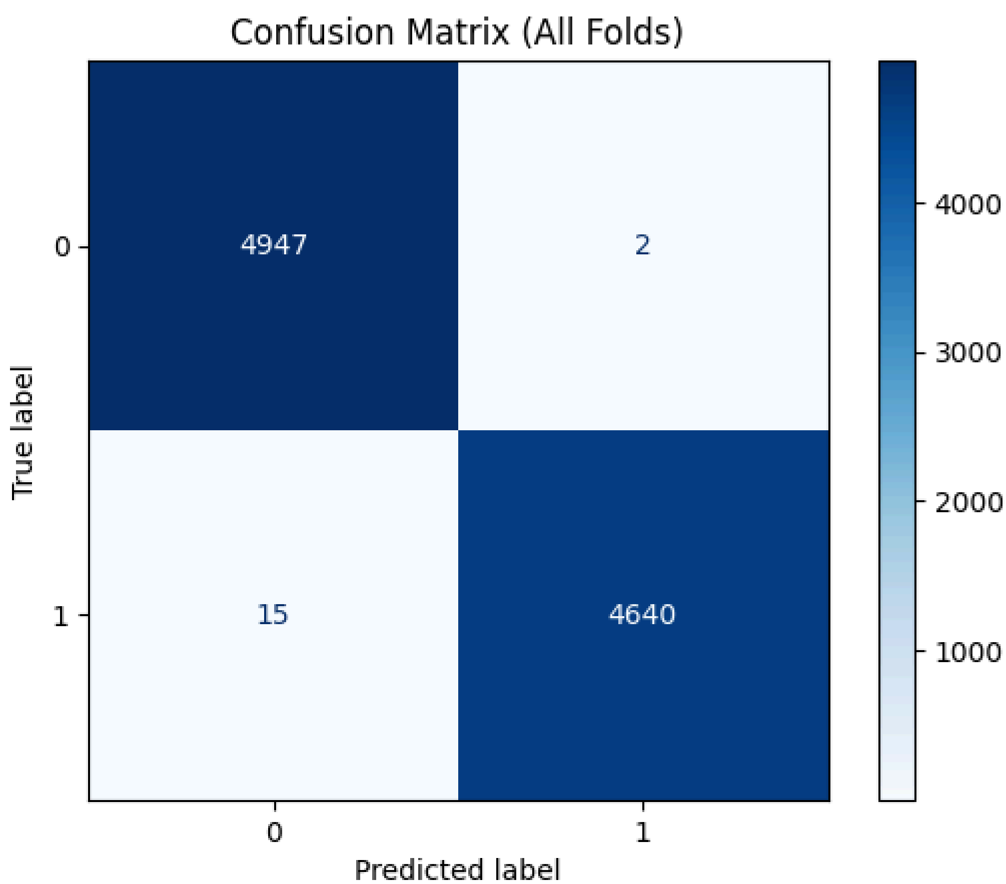
```
Final CNN Cross-Validation Results:
Fold Accuracies: [1.0, 0.9989588856697083, 0.9963560700416565, 0.9963560700416565,
0.7770833373069763]
Mean Accuracy: 0.9538
Standard Deviation: 0.0883
```

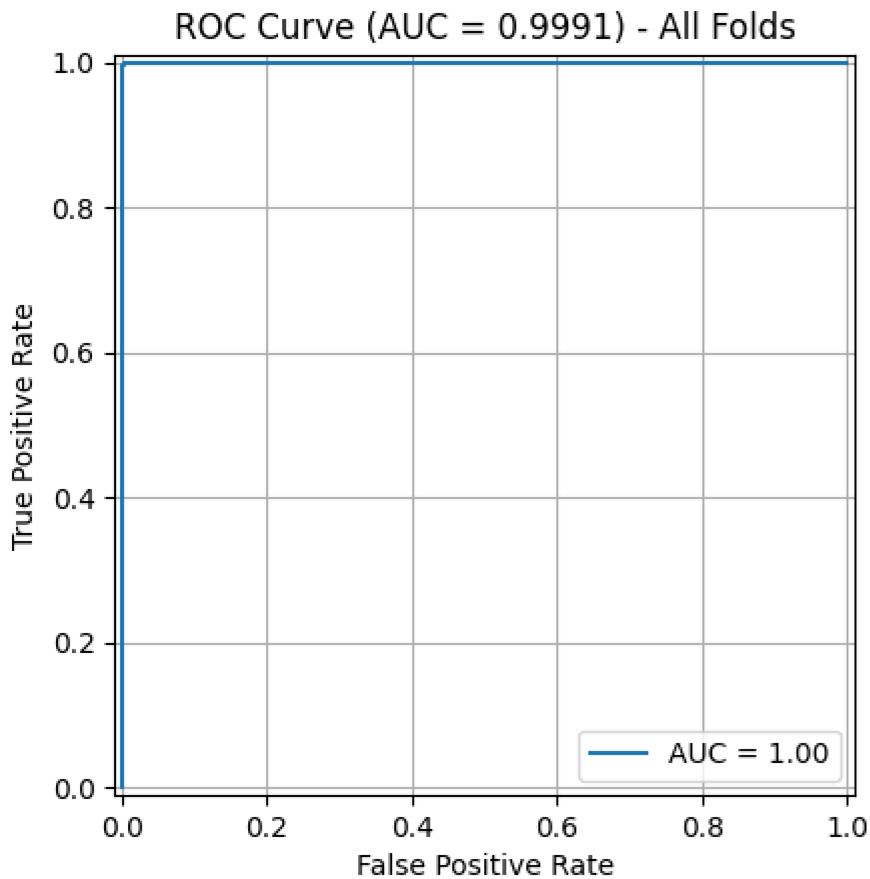
Step 5: Visual Evaluation

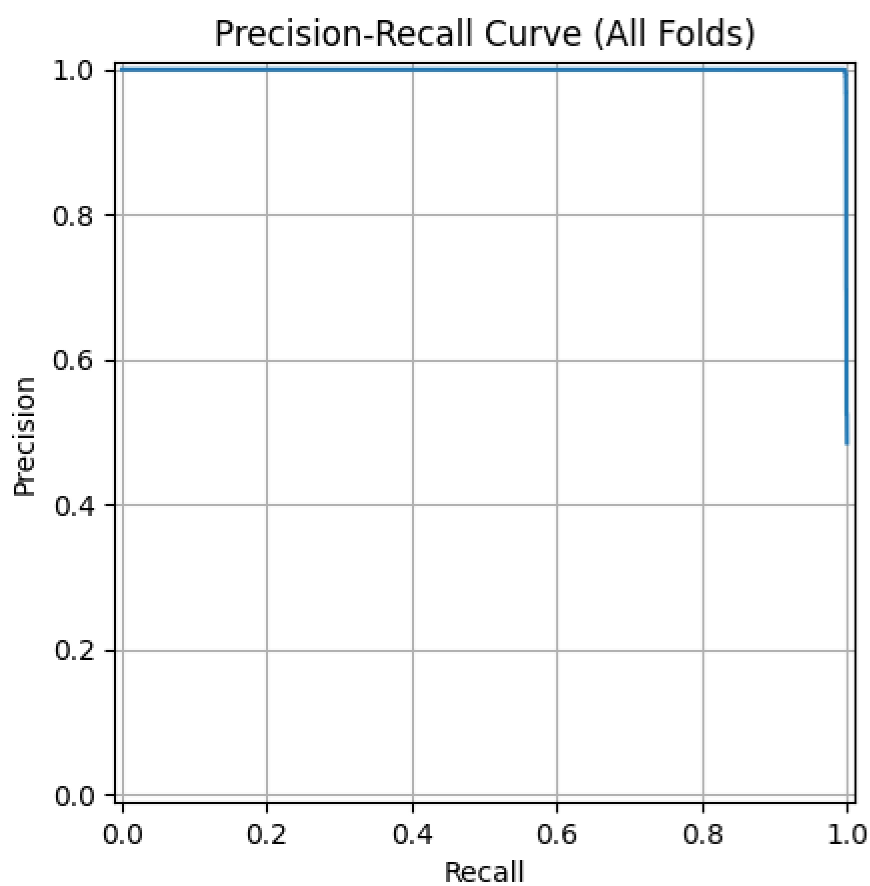
In [5]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    roc_curve,
    auc,
    RocCurveDisplay,
    precision_recall_curve,
    PrecisionRecallDisplay
)

# Confusion Matrix
cm = confusion_matrix(all_y_true, all_y_pred)
disp_cm = ConfusionMatrixDisplay(confusion_matrix=cm)
disp_cm.plot(cmap='Blues')
plt.title('Confusion Matrix (All Folds)')
```

```
plt.grid(False)
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(all_y_true, all_y_scores)
roc_auc = auc(fpr, tpr)
RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc).plot()
plt.title(f'ROC Curve (AUC = {roc_auc:.4f}) - All Folds')
plt.grid(True)
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(all_y_true, all_y_scores)
PrecisionRecallDisplay(precision=precision, recall=recall).plot()
plt.title('Precision-Recall Curve (All Folds)')
plt.grid(True)
plt.show()
```




ROC Curve (AUC = 0.9991) - All Folds
True Positive Rate
False Positive Rate
AUC = 1.00
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve (All Folds)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

saving the model as PDF

In [11]:
```
import os
os.getcwd()
```

Out[11]:
```
'd:\\Coding Projects\\Detection-of-SYN-Flood-Attacks-Using-Machine-Learning-and-Deep-Learning-Techniques-with-Feature-Base\\Amir Tavahin'
```

In [ ]:
```
!jupyter nbconvert --to webpdf "d:\\Coding Projects\\Detection-of-SYN-Flood-Attacks
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include
the error message in the cell output (the default behaviour is to abort conversion).
This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with defaul
t basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
mat=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
mat=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each cel
l).
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_for
```

```
mat=notebook --FilesWriter.build_directory= --CoalesceStreamsPreprocessor.enabled=Tr
ue]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.
exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporte
r.exclude_input=True --TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the sys
tem.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful for
the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITI
CAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
```

```
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook. To re
cover
                                  previous default behaviour (outputting to the curr
ent
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-sl
ideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------
```

```
    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.

[NbConvertApp] WARNING | pattern 'Amir Tavahin/CNN.ipynb' matched no files
```